US009473545B2

(12) **United States Patent**
Archer et al.

(10) **Patent No.: US 9,473,545 B2**
(45) **Date of Patent: Oct. 18, 2016**

(54) ADMINISTERING GROUP IDENTIFIERS OF PROCESSES IN A PARALLEL COMPUTER

(71) Applicant: **LENOVO ENTERPRISE SOLUTIONS (SINGAPORE) PTE. LTD.**, Singapore (SG)

(72) Inventors: **Charles J. Archer**, Rochester, MN (US); **Tsai-Yang Jea**, Poughkeepsie, NY (US); **Chulho Kim**, Poughkeepsie, NY (US)

(73) Assignee: **Lenovo Enterprise Solutions (Singapore) Pte. Ltd.**, Singapore (SG)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 547 days.

(21) Appl. No.: **14/026,107**

(22) Filed: **Sep. 13, 2013**

(65) **Prior Publication Data**

US 2015/0081862 A1 Mar. 19, 2015

(51) **Int. Cl.**
H04L 29/06 (2006.01)
G06F 9/50 (2006.01)

(52) **U.S. Cl.**
CPC ***H04L 65/4023*** (2013.01); ***G06F 9/5066*** (2013.01); *G06F 2209/5011* (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,012,090	A *	1/2000	Chung	H04L 67/02 709/218
7,584,342	B1 *	9/2009	Nordquist	G06F 9/3851 712/22
2004/0215835	A1 *	10/2004	Liu	G06F 11/221 710/1
2009/0292744	A1 *	11/2009	Matsumura	G06F 17/30575
2009/0307699	A1 *	12/2009	Munshi	G06F 9/4843 718/102
2009/0307704	A1 *	12/2009	Munshi	G06F 9/4843 718/104
2015/0324444	A1 *	11/2015	Chercoles Sanchez	G06F 17/30545 707/703

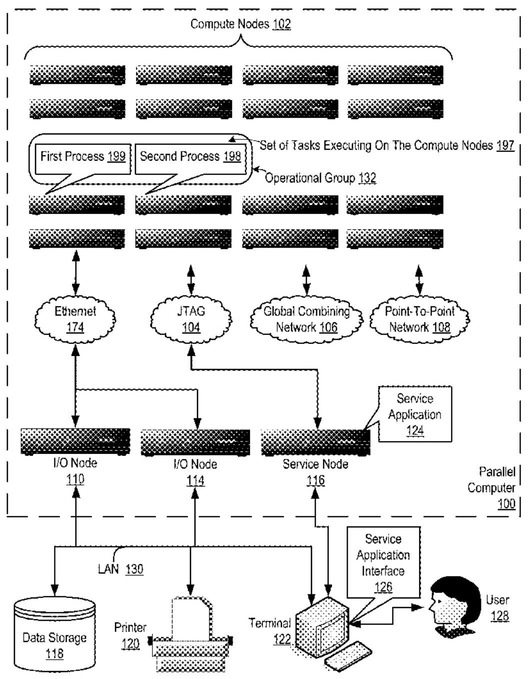
* cited by examiner

Primary Examiner — Natisha Cox
(74) *Attorney, Agent, or Firm* — Brandon C. Kennedy; Katherine S. Brown; Kennedy Lenart Spraggins LLP

(57) **ABSTRACT**

Administering group identifiers of processes in a parallel computer includes each process in a set of processes, receiving from a compute node of the plurality of compute nodes, a request to establish the set of processes as an operational group including receiving a list of process identifiers for each process of the set of processes. Embodiments also include each process generating without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers.

20 Claims, 12 Drawing Sheets



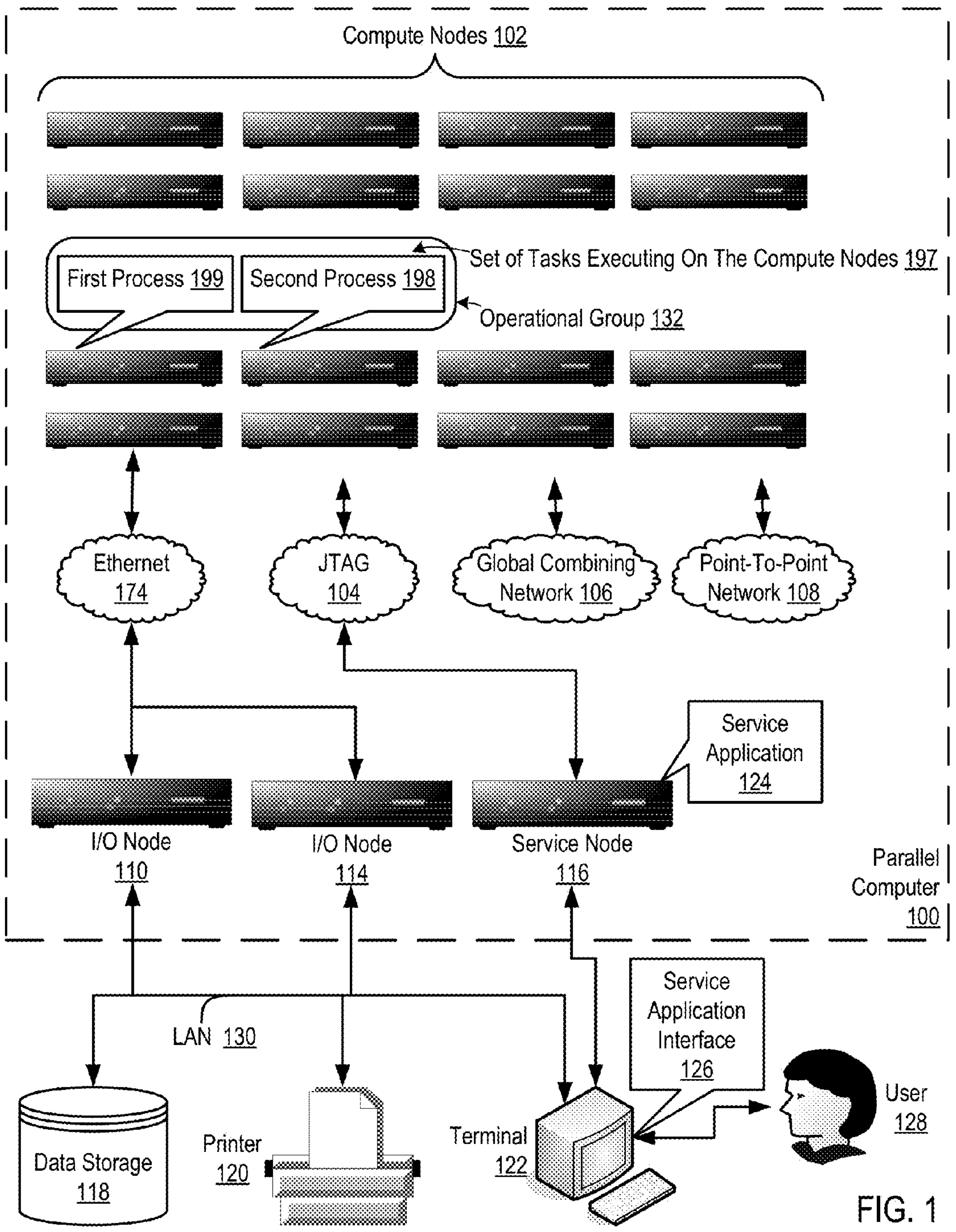

Compute Nodes 102
Set of Tasks Executing On The Compute Nodes 197
First Process 199
Second Process 198
Operational Group 132
Ethernet 174
JTAG 104
Global Combining Network 106
Point-To-Point Network 108
Service Application 124
I/O Node 110
I/O Node 114
Service Node 116
Parallel Computer 100
LAN 130
Service Application Interface 126
User 128
Data Storage 118
Printer 120
Terminal 122

FIG. 1

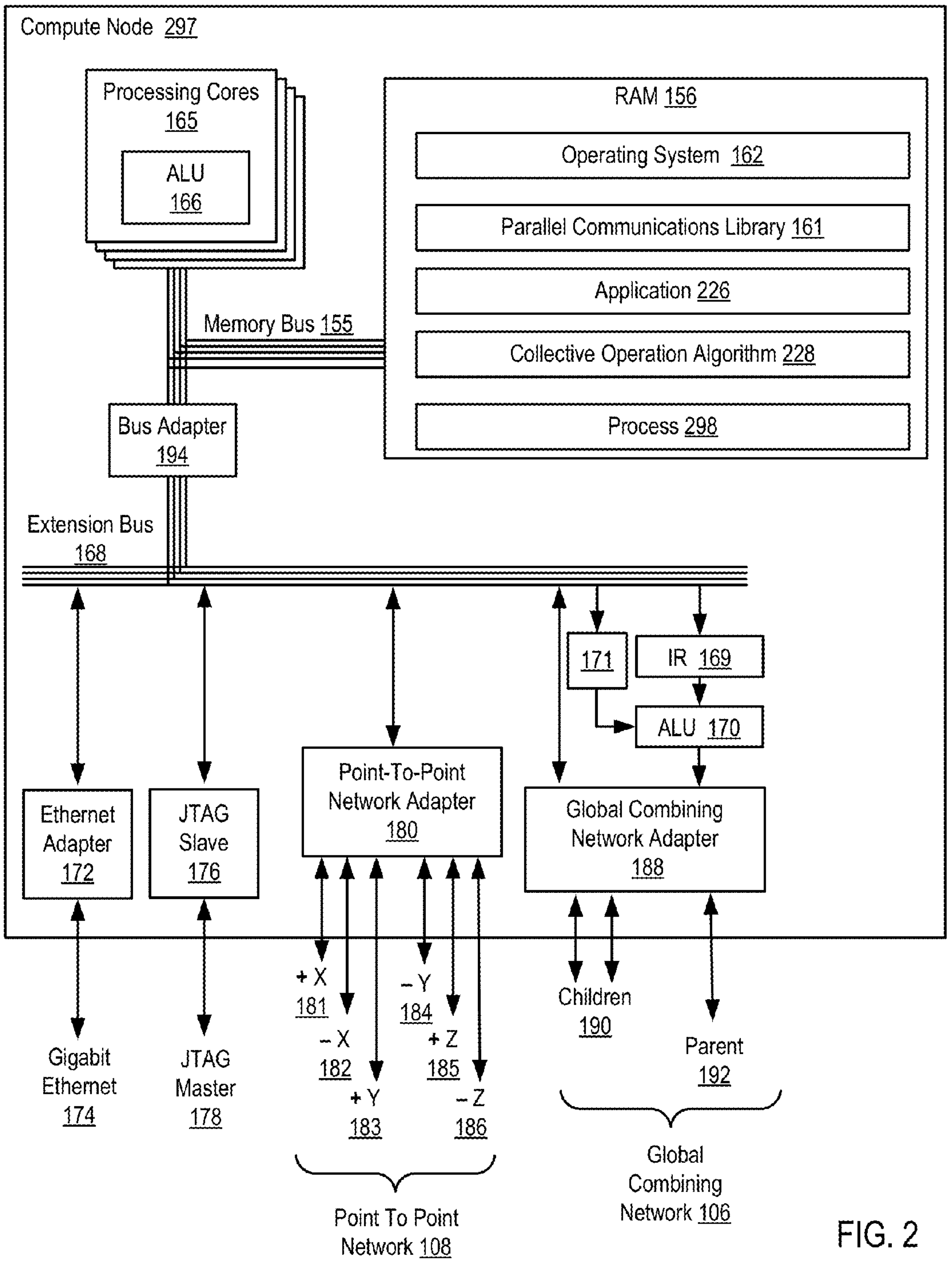

Compute Node 297
Processing Cores 165
ALU 166
RAM 156
Operating System 162
Parallel Communications Library 161
Application 226
Collective Operation Algorithm 228
Process 298
Memory Bus 155
Bus Adapter 194
Extension Bus 168
171
IR 169
ALU 170
Ethernet Adapter 172
JTAG Slave 176
Point-To-Point Network Adapter 180
Global Combining Network Adapter 188
+X 181
−X 182
+Y 183
−Y 184
+Z 185
−Z 186
Children 190
Parent 192
Gigabit Ethernet 174
JTAG Master 178
Point To Point Network 108
Global Combining Network 106

FIG. 2

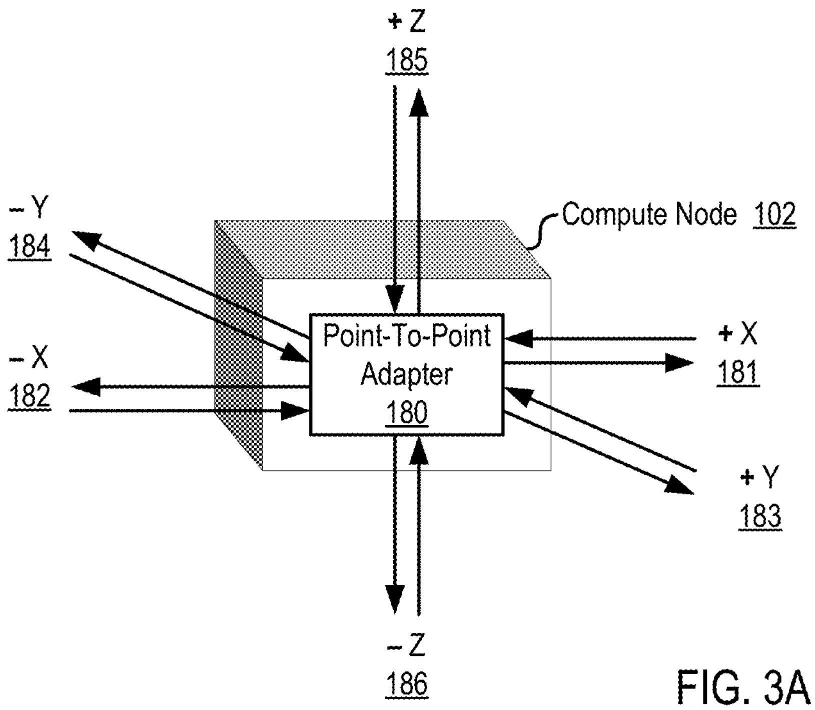

+ Z
185
Compute Node 102
– Y
184
Point-To-Point Adapter 180
+ X
181
– X
182
+ Y
183
– Z
186

FIG. 3A

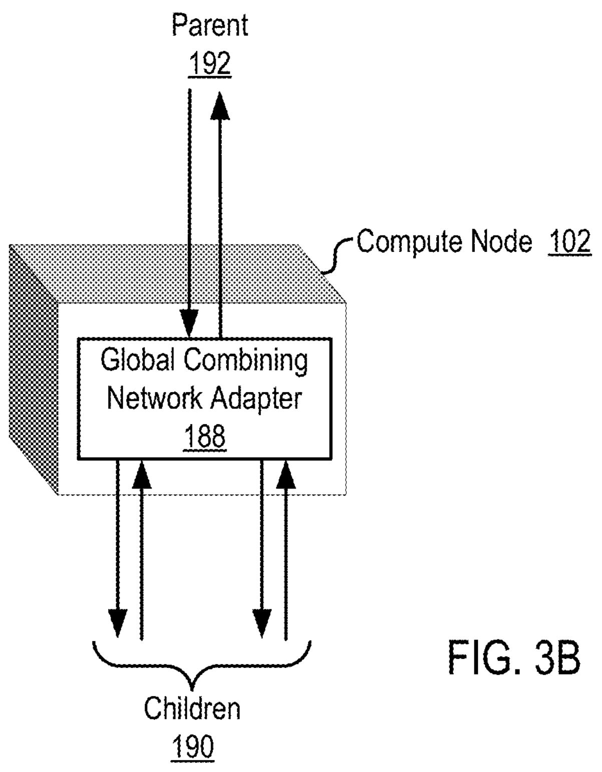

Parent
192
Compute Node 102
Global Combining Network Adapter
188
Children
190

FIG. 3B

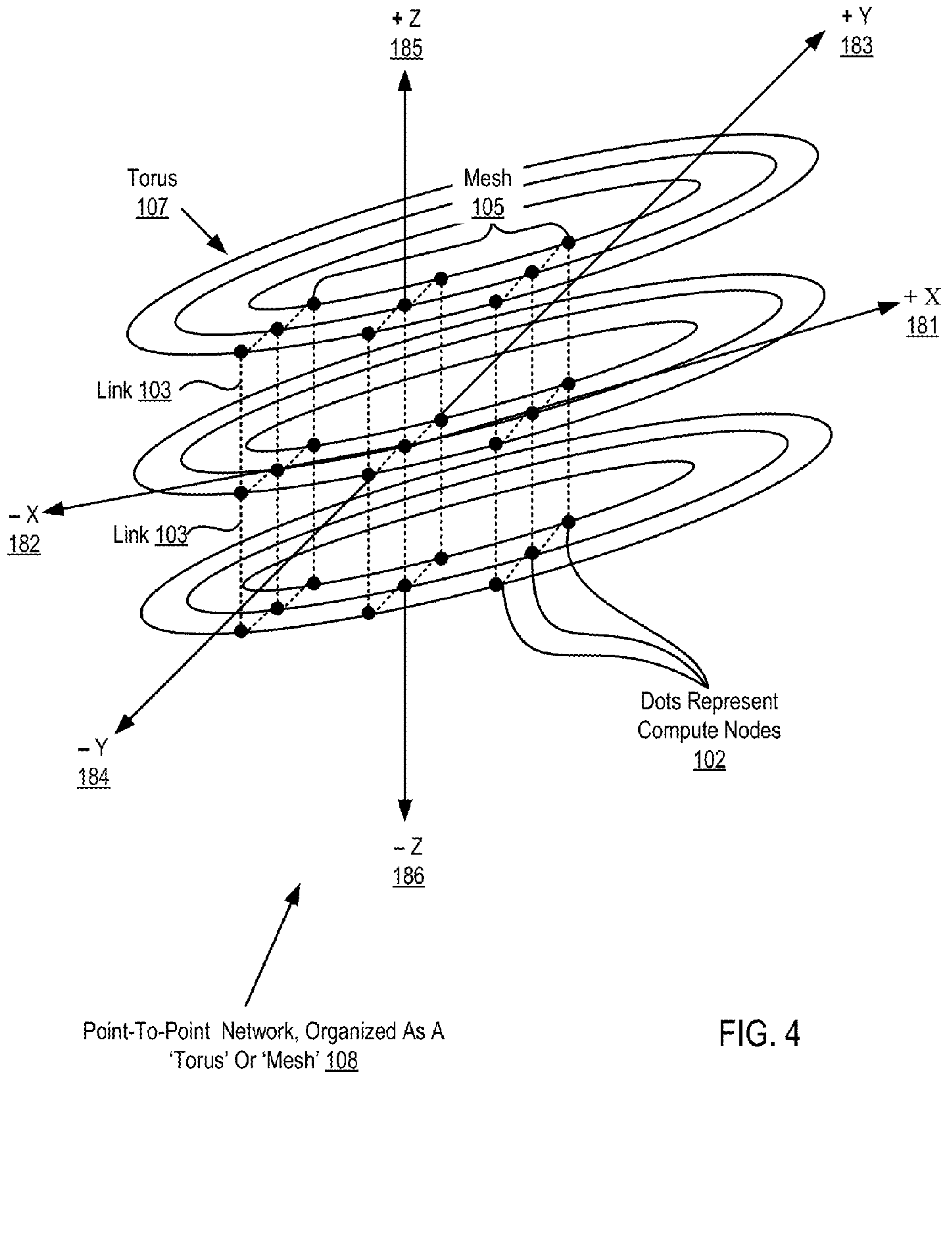

+ Z
185
+ Y
183
Torus
107
Mesh
105
+ X
181
Link 103
Link 103
- X
182
Dots Represent
Compute Nodes
102
- Y
184
- Z
186
Point-To-Point Network, Organized As A 'Torus' Or 'Mesh' 108

FIG. 4

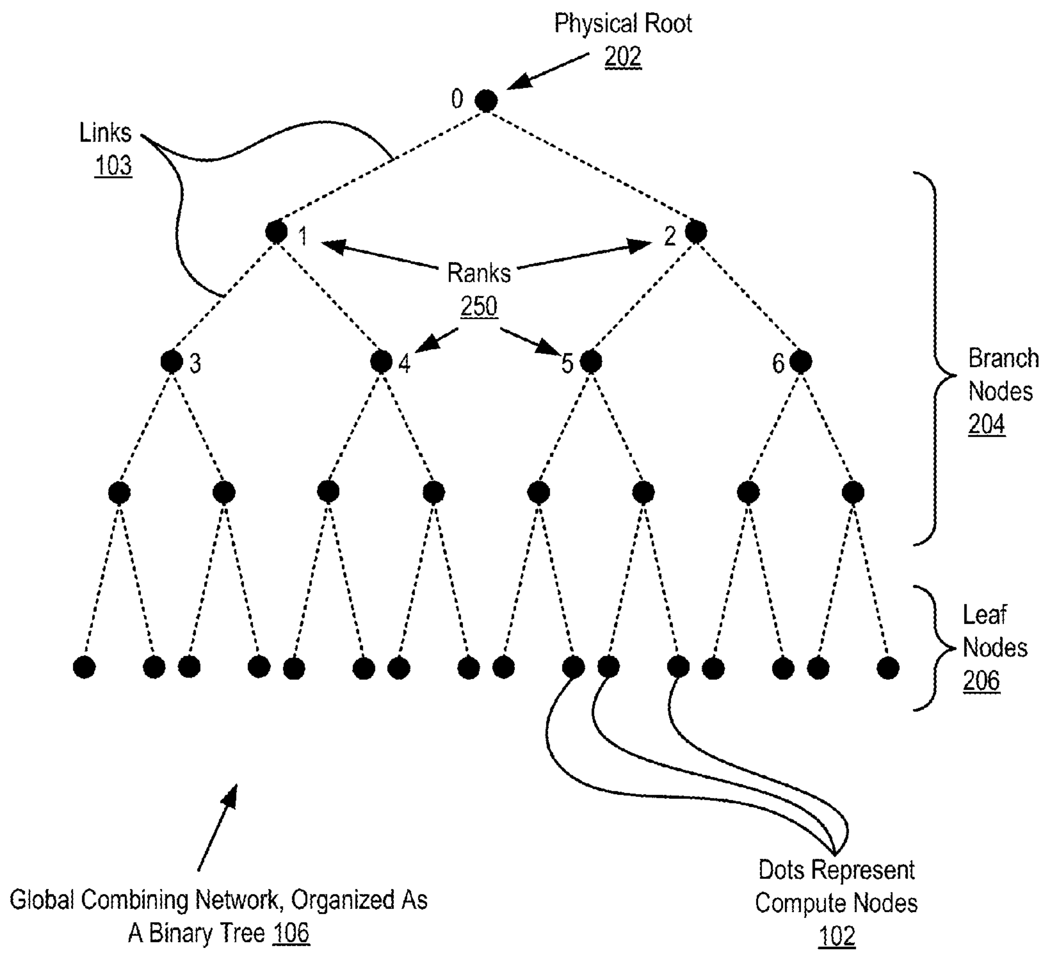

Physical Root 202
0
Links 103
1
2
Ranks 250
3
4
5
6
Branch Nodes 204
Leaf Nodes 206
Global Combining Network, Organized As A Binary Tree 106
Dots Represent Compute Nodes 102

FIG. 5

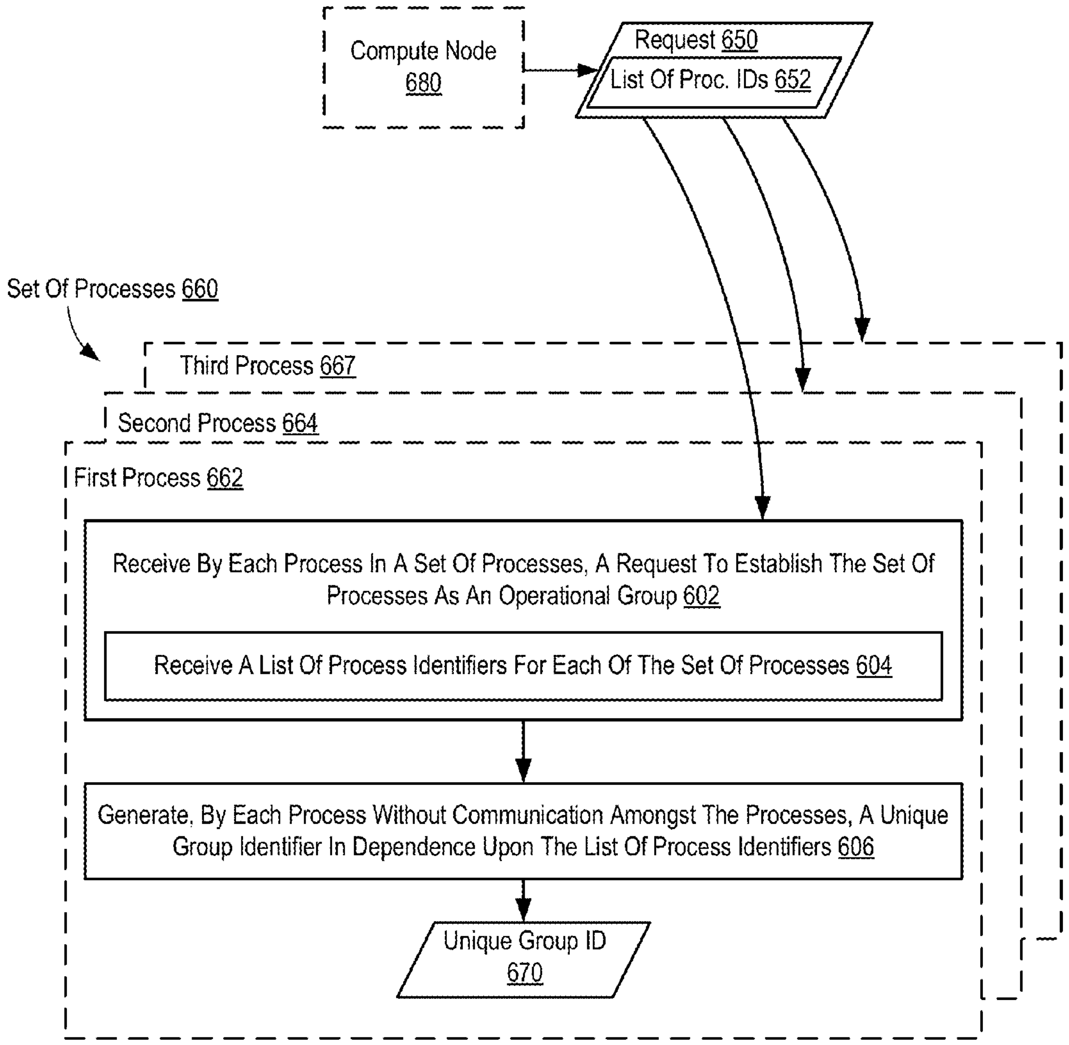

Compute Node 680
Request 650
List Of Proc. IDs 652
Set Of Processes 660
Third Process 667
Second Process 664
First Process 662
Receive By Each Process In A Set Of Processes, A Request To Establish The Set Of Processes As An Operational Group 602
Receive A List Of Process Identifiers For Each Of The Set Of Processes 604
Generate, By Each Process Without Communication Amongst The Processes, A Unique Group Identifier In Dependence Upon The List Of Process Identifiers 606
Unique Group ID 670

FIG. 6

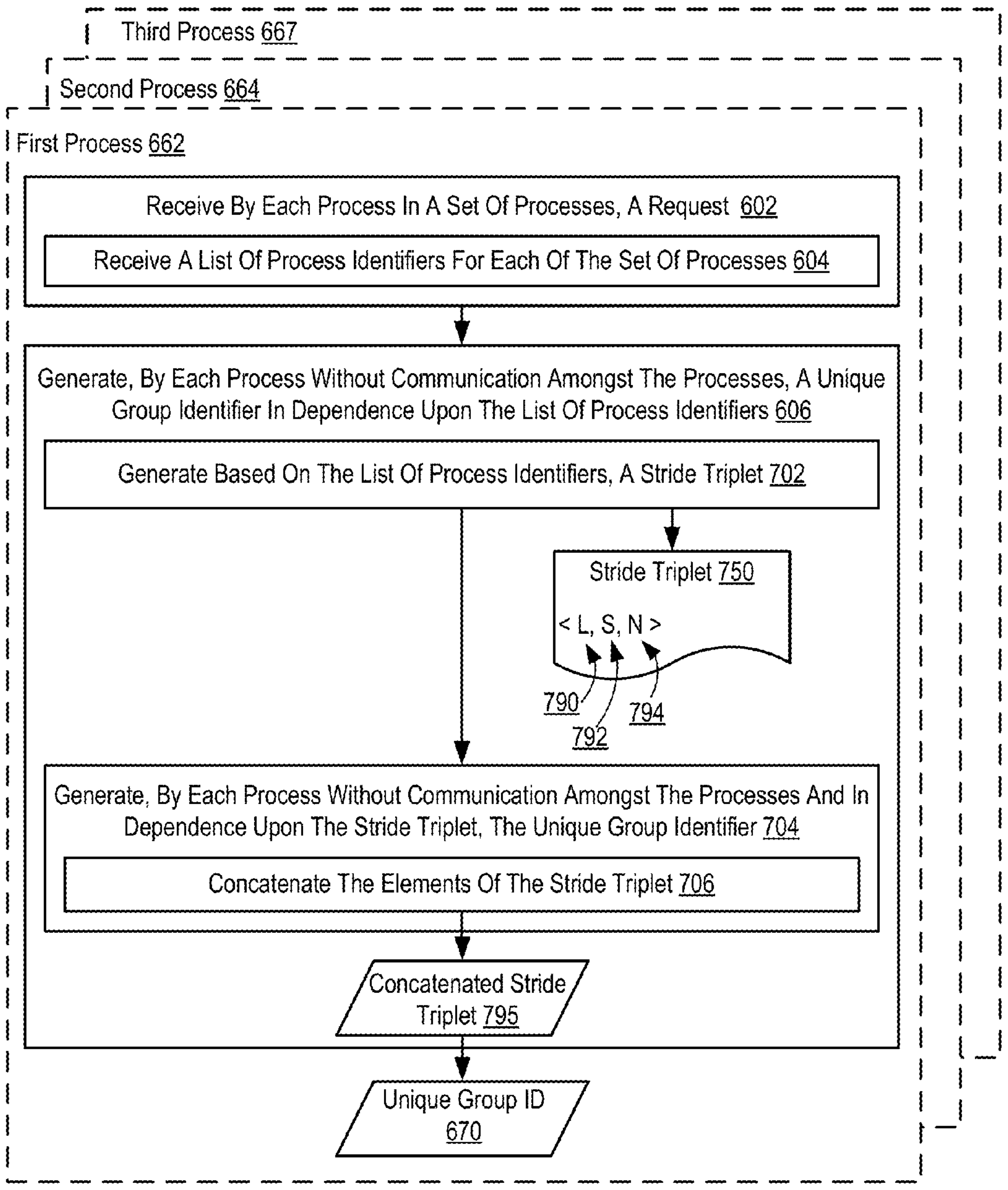
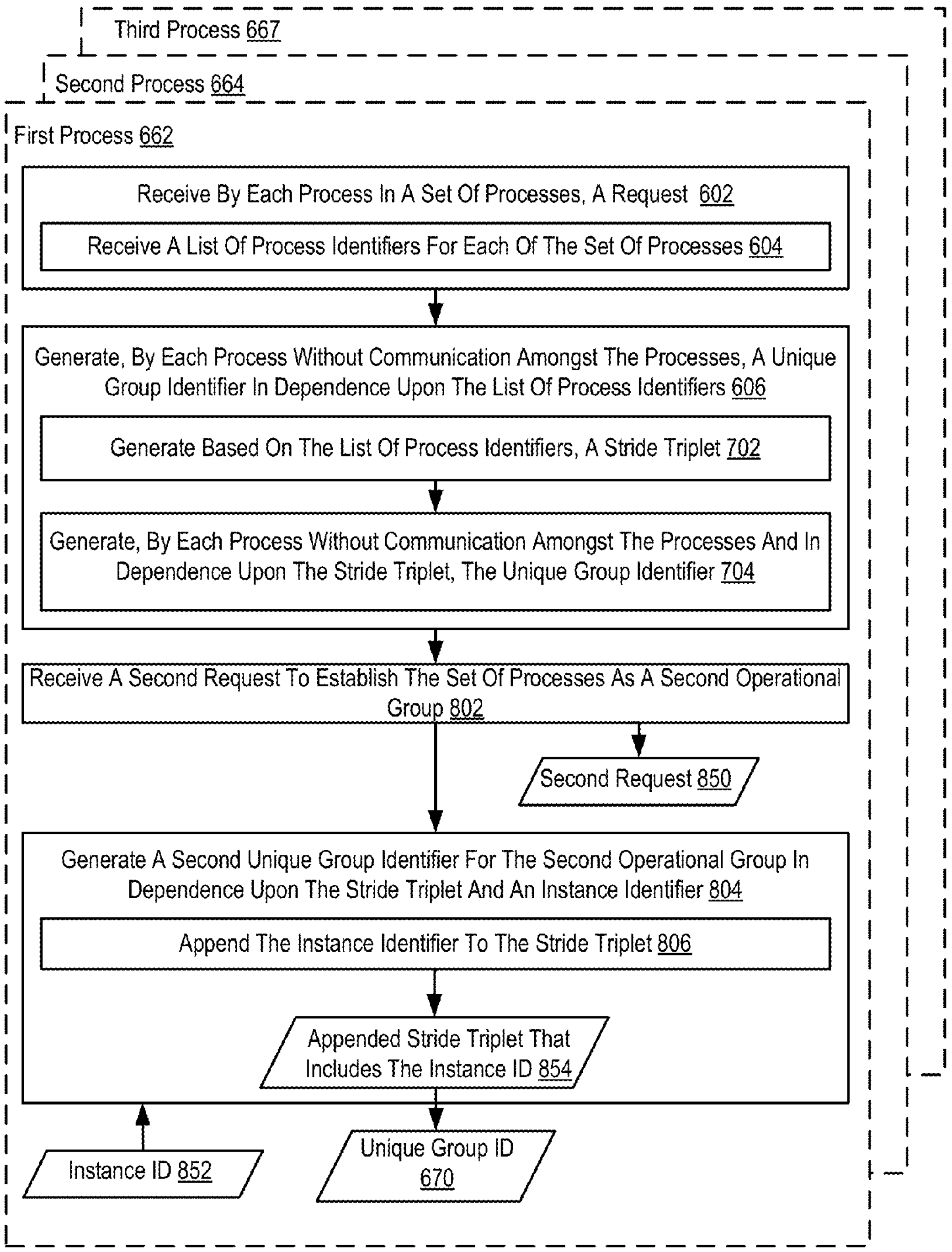

Third Process 667
Second Process 664
First Process 662
Receive By Each Process In A Set Of Processes, A Request 602
Receive A List Of Process Identifiers For Each Of The Set Of Processes 604
Generate, By Each Process Without Communication Amongst The Processes, A Unique Group Identifier In Dependence Upon The List Of Process Identifiers 606
Generate Based On The List Of Process Identifiers, A Stride Triplet 702
Stride Triplet 750
< L, S, N >
790
792
794
Generate, By Each Process Without Communication Amongst The Processes And In Dependence Upon The Stride Triplet, The Unique Group Identifier 704
Concatenate The Elements Of The Stride Triplet 706
Concatenated Stride Triplet 795
Unique Group ID 670

FIG. 7



Third Process 667
Second Process 664
First Process 662
Receive By Each Process In A Set Of Processes, A Request 602
Receive A List Of Process Identifiers For Each Of The Set Of Processes 604
Generate, By Each Process Without Communication Amongst The Processes, A Unique Group Identifier In Dependence Upon The List Of Process Identifiers 606
Generate Based On The List Of Process Identifiers, A Stride Triplet 702
Generate, By Each Process Without Communication Amongst The Processes And In Dependence Upon The Stride Triplet, The Unique Group Identifier 704
Receive A Second Request To Establish The Set Of Processes As A Second Operational Group 802
Second Request 850
Generate A Second Unique Group Identifier For The Second Operational Group In Dependence Upon The Stride Triplet And An Instance Identifier 804
Append The Instance Identifier To The Stride Triplet 806
Appended Stride Triplet That Includes The Instance ID 854
Instance ID 852
Unique Group ID 670

FIG. 8

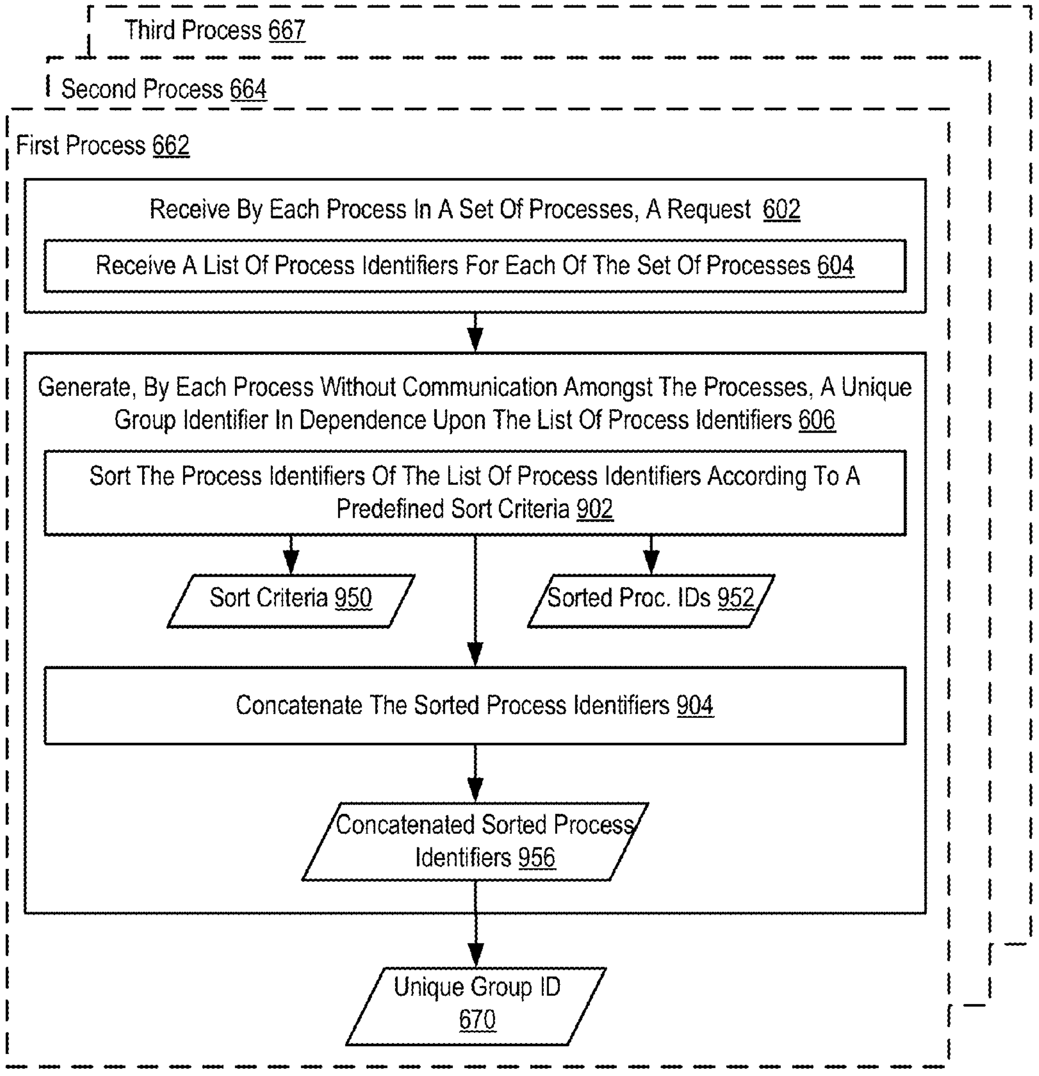

Third Process 667
Second Process 664
First Process 662
Receive By Each Process In A Set Of Processes, A Request 602
Receive A List Of Process Identifiers For Each Of The Set Of Processes 604
Generate, By Each Process Without Communication Amongst The Processes, A Unique Group Identifier In Dependence Upon The List Of Process Identifiers 606
Sort The Process Identifiers Of The List Of Process Identifiers According To A Predefined Sort Criteria 902
Sort Criteria 950
Sorted Proc. IDs 952
Concatenate The Sorted Process Identifiers 904
Concatenated Sorted Process Identifiers 956
Unique Group ID 670

FIG. 9

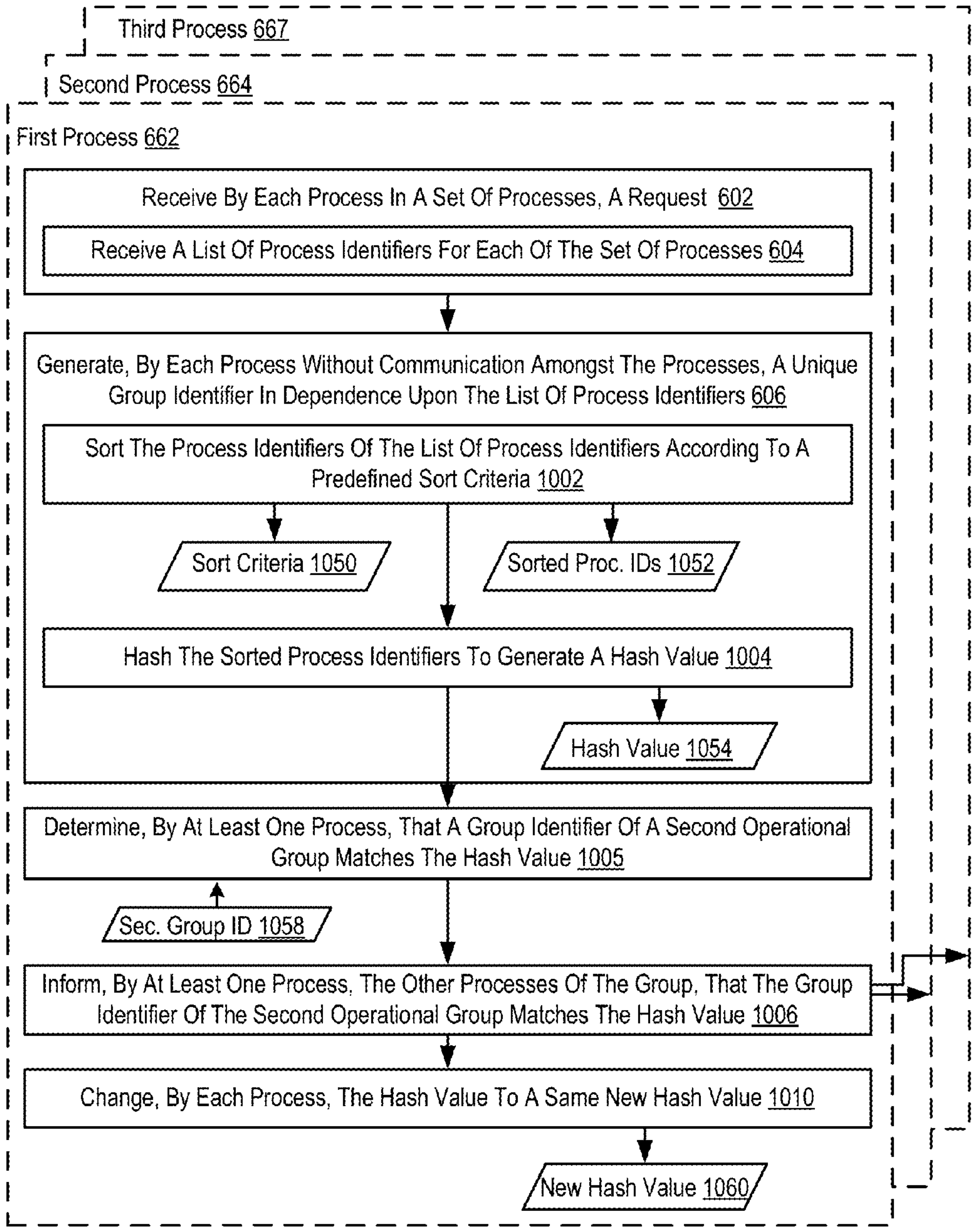

Third Process 667
Second Process 664
First Process 662
Receive By Each Process In A Set Of Processes, A Request 602
Receive A List Of Process Identifiers For Each Of The Set Of Processes 604
Generate, By Each Process Without Communication Amongst The Processes, A Unique Group Identifier In Dependence Upon The List Of Process Identifiers 606
Sort The Process Identifiers Of The List Of Process Identifiers According To A Predefined Sort Criteria 1002
Sort Criteria 1050
Sorted Proc. IDs 1052
Hash The Sorted Process Identifiers To Generate A Hash Value 1004
Hash Value 1054
Determine, By At Least One Process, That A Group Identifier Of A Second Operational Group Matches The Hash Value 1005
Sec. Group ID 1058
Inform, By At Least One Process, The Other Processes Of The Group, That The Group Identifier Of The Second Operational Group Matches The Hash Value 1006
Change, By Each Process, The Hash Value To A Same New Hash Value 1010
New Hash Value 1060

FIG. 10

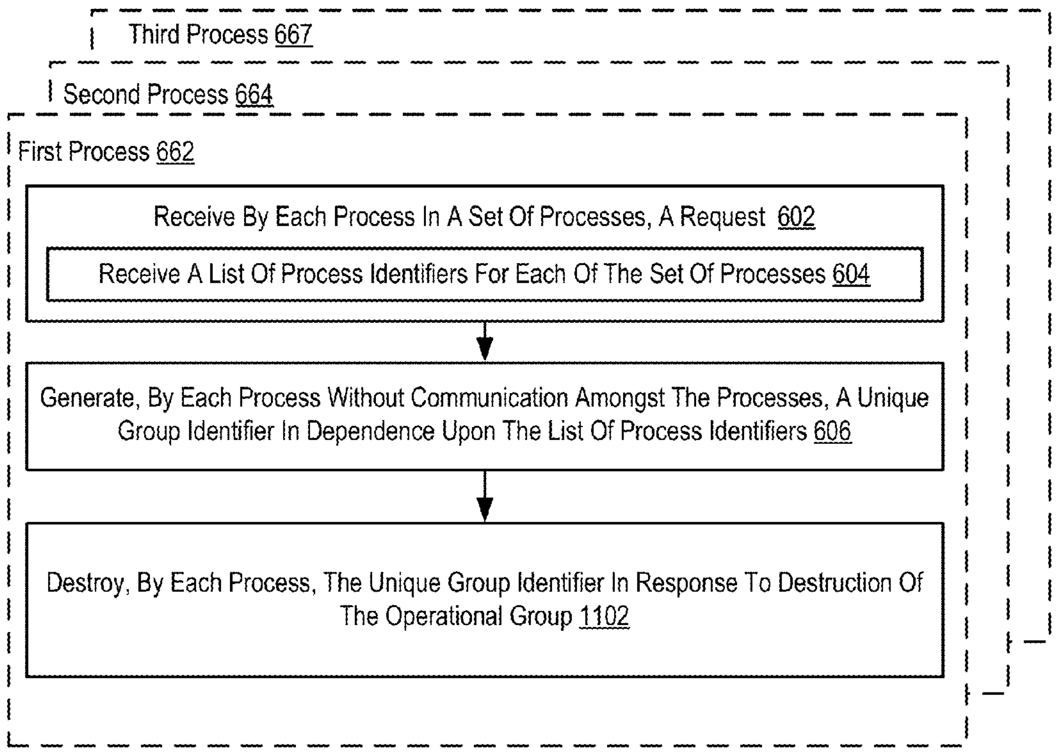

Third Process 667
Second Process 664
First Process 662
Receive By Each Process In A Set Of Processes, A Request 602
Receive A List Of Process Identifiers For Each Of The Set Of Processes 604
Generate, By Each Process Without Communication Amongst The Processes, A Unique Group Identifier In Dependence Upon The List Of Process Identifiers 606
Destroy, By Each Process, The Unique Group Identifier In Response To Destruction Of The Operational Group 1102

FIG. 11

ADMINISTERING GROUP IDENTIFIERS OF PROCESSES IN A PARALLEL COMPUTER

BACKGROUND OF THE INVENTION

1. Field of the Invention

The field of the invention is data processing, or, more specifically, methods, apparatus, and products for administering group identifiers of processes in a parallel computer.

2. Description of Related Art

The development of the EDVAC computer system of 1948 is often cited as the beginning of the computer era. Since that time, computer systems have evolved into extremely complicated devices. Today's computers are much more sophisticated than early systems such as the EDVAC. Computer systems typically include a combination of hardware and software components, application programs, operating systems, processors, buses, memory, input/output devices, and so on. As advances in semiconductor processing and computer architecture push the performance of the computer higher and higher, more sophisticated computer software has evolved to take advantage of the higher performance of the hardware, resulting in computer systems today that are much more powerful than just a few years ago.

Parallel computing is an area of computer technology that has experienced advances. Parallel computing is the simultaneous execution of the same task (split up and specially adapted) on multiple processors in order to obtain results faster. Parallel computing is based on the fact that the process of solving a problem usually can be divided into smaller tasks, which may be carried out simultaneously with some coordination.

Parallel computers execute parallel algorithms. A parallel algorithm can be split up to be executed a piece at a time on many different processing devices, and then put back together again at the end to get a data processing result. Some algorithms are easy to divide up into pieces. Splitting up the job of checking all of the numbers from one to a hundred thousand to see which are primes could be done, for example, by assigning a subset of the numbers to each available processor, and then putting the list of positive results back together. In this specification, the multiple processing devices that execute the individual pieces of a parallel program are referred to as 'compute nodes.' A parallel computer is composed of compute nodes and other processing nodes as well, including, for example, input/output ('I/O') nodes, and service nodes.

Parallel algorithms are valuable because it is faster to perform some kinds of large computing tasks via a parallel algorithm than it is via a serial (non-parallel) algorithm, because of the way modern processors work. It is far more difficult to construct a computer with a single fast processor than one with many slow processors with the same throughput. There are also certain theoretical limits to the potential speed of serial processors. On the other hand, every parallel algorithm has a serial part and so parallel algorithms have a saturation point. After that point adding more processors does not yield any more throughput but only increases the overhead and cost.

Parallel algorithms are designed also to optimize one more resource the data communications requirements among the nodes of a parallel computer. There are two ways parallel processors communicate, shared memory or message passing. Shared memory processing needs additional

locking for the data and imposes the overhead of additional processor and bus cycles and also serializes some portion of the algorithm.

Message passing processing uses high-speed data communications networks and message buffers, but this communication adds transfer overhead on the data communications networks as well as additional memory need for message buffers and latency in the data communications among nodes. Designs of parallel computers use specially designed data communications links so that the communication overhead will be small but it is the parallel algorithm that decides the volume of the traffic.

Many data communications network architectures are used for message passing among nodes in parallel computers. Compute nodes may be organized in a network as a 'torus' or 'mesh,' for example. Also, compute nodes may be organized in a network as a tree. A torus network connects the nodes in a three-dimensional mesh with wrap around links. Every node is connected to its six neighbors through this torus network, and each node is addressed by its x,y,z coordinate in the mesh. In such a manner, a torus network lends itself to point to point operations. In a tree network, the nodes typically are connected into a binary tree: each node has a parent, and two children (although some nodes may only have zero children or one child, depending on the hardware configuration). Although a tree network typically is inefficient in point to point communication, a tree network does provide high bandwidth and low latency for certain collective operations, message passing operations where all compute nodes participate simultaneously, such as, for example, an allgather operation. In computers that use a torus and a tree network, the two networks typically are implemented independently of one another, with separate routing circuits, separate physical links, and separate message buffers.

SUMMARY OF THE INVENTION

Administering group identifiers of processes in a parallel computer includes each process in a set of processes, receiving from a compute node of the plurality of compute nodes, a request to establish the set of processes as an operational group including receiving a list of process identifiers for each process of the set of processes. Embodiments also include each process generating without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers.

The foregoing and other objects, features and advantages of the invention will be apparent from the following more particular descriptions of exemplary embodiments of the invention as illustrated in the accompanying drawings wherein like reference numbers generally represent like parts of exemplary embodiments of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates an exemplary system for administering group identifiers of processes in a parallel computer according to embodiments of the present invention.

FIG. **2** sets forth a block diagram of an example compute node useful in a parallel computer capable of administering group identifiers of processes according to embodiments of the present invention.

FIG. **3**A sets forth a block diagram of an example Point-To-Point Adapter useful in systems for administering group identifiers of processes in a parallel computer according to embodiments of the present invention.

FIG. **3**B sets forth a block diagram of an example Global Combining Network Adapter useful in systems for administering group identifiers of processes in a parallel computer according to embodiments of the present invention.

FIG. **4** sets forth a line drawing illustrating an example data communications network optimized for point-to-point operations useful in systems capable of administering group identifiers of processes in a parallel computer according to embodiments of the present invention.

FIG. **5** sets forth a line drawing illustrating an example global combining network useful in systems capable of administering group identifiers of processes in a parallel computer according to embodiments of the present invention.

FIG. **6** sets forth a flow chart illustrating an example method for administering group identifiers of processes in a parallel computer according to embodiments of the present invention.

FIG. **7** sets forth a flow chart illustrating another example method for administering group identifiers of processes in a parallel computer according to embodiments of the present invention.

FIG. **8** sets forth a flow chart illustrating another example method for administering group identifiers of processes in a parallel computer according to embodiments of the present invention.

FIG. **9** sets forth a flow chart illustrating another example method for administering group identifiers of processes in a parallel computer according to embodiments of the present invention.

FIG. **10** sets forth a flow chart illustrating another example method for administering group identifiers of processes in a parallel computer according to embodiments of the present invention.

FIG. **11** sets forth a flow chart illustrating another example method for administering group identifiers of processes in a parallel computer according to embodiments of the present invention.

DETAILED DESCRIPTION OF EXEMPLARY EMBODIMENTS

Exemplary methods, apparatuses, and computer program products for administering group identifiers of processes in a parallel computer in accordance with the present invention are described with reference to the accompanying drawings, beginning with FIG. **1**. FIG. **1** illustrates an exemplary system for administering group identifiers of processes in a parallel computer according to embodiments of the present invention. The system of FIG. **1** includes a parallel computer (**100**), non-volatile memory for the computer in the form of a data storage device (**118**), an output device for the computer in the form of a printer (**120**), and an input/output device for the computer in the form of a computer terminal (**122**).

The parallel computer (**100**) in the example of FIG. **1** includes a plurality of compute nodes (**102**). The compute nodes (**102**) are coupled for data communications by several independent data communications networks including a high speed Ethernet network (**174**), a Joint Test Action Group ('JTAG') network (**104**), a global combining network (**106**) which is optimized for collective operations using a binary tree network topology, and a point-to-point network (**108**), which is optimized for point-to-point operations using a torus network topology. The global combining network (**106**) is a data communications network that includes data communications links connected to the compute nodes (**102**) so as to organize the compute nodes (**102**) as a binary tree. Each data communications network is implemented with data communications links among the compute nodes (**102**). The data communications links provide data communications for parallel operations among the compute nodes (**102**) of the parallel computer (**100**).

The compute nodes (**102**) of the parallel computer (**100**) include a set (**197**) of processes including a first process (**199**) and a second process (**198**). The first process (**199**) and the second process (**198**) are organized into an operational group (**132**) for collective parallel operations on the parallel computer (**100**). A set of process may include any number of processes and each process may be a member of any number of operational groups. Each operational group (**132**) of processes is the set of processes that execute a collective parallel operation. Each process in the operational group (**132**) is assigned a unique rank or process identifier that identifies the particular process in the operational group (**132**). Collective operations are implemented with data communications among the processes of an operational group. Collective operations are those functions that involve all the processes of an operational group (**132**). A collective operation is an operation, a message-passing computer program instruction that is executed simultaneously, that is, at approximately the same time, by all the processes in an operational group (**132**) of processes. Such an operational group (**132**) may include all the processes executing on the compute nodes (**102**) in a parallel computer (**100**) or a subset of all the processes executing on the compute nodes (**102**). Collective operations are often built around point-to-point operations. A collective operation requires that all processes within an operational group (**132**) call the same collective operation with matching arguments. A 'broadcast' is an example of a collective operation for moving data among processes of an operational group. A 'reduce' operation is an example of a collective operation that executes arithmetic or logical functions on data distributed among the processes of an operational group (**132**). An operational group (**132**) may be implemented as, for example, an MPI 'communicator.'

'MPI' refers to 'Message Passing Interface,' a prior art parallel communications library, a module of computer program instructions for data communications on parallel computers. Examples of prior-art parallel communications libraries that may be improved for use in systems configured according to embodiments of the present invention include MPI and the 'Parallel Virtual Machine' ('PVM') library. PVM was developed by the University of Tennessee, The Oak Ridge National Laboratory and Emory University. MPI is promulgated by the MPI Forum, an open group with representatives from many organizations that define and maintain the MPI standard. MPI at the time of this writing is a de facto standard for communication among compute nodes running a parallel program on a distributed memory parallel computer. This specification sometimes uses MPI terminology for ease of explanation, although the use of MPI as such is not a requirement or limitation of the present invention.

Some collective operations have a single originating or receiving process in an operational group (**132**). For example, in a 'broadcast' collective operation, the process that distributes the data to all the other processes is an originating process. In a 'gather' operation, for example, the process that received all the data from the other processes is a receiving process. The originating or receiving process is referred to as a logical root.

Most collective operations are variations or combinations of four basic operations: broadcast, gather, scatter, and reduce. The interfaces for these collective operations are defined in the MPI standards promulgated by the MPI Forum. Algorithms for executing collective operations, however, are not defined in the MPI standards. In a broadcast operation, all processes specify the same root process, whose buffer contents will be sent. Processes other than the root specify receive buffers. After the operation, all buffers contain the message from the root process.

A scatter operation, like the broadcast operation, is also a one-to-many collective operation. In a scatter operation, the logical root divides data on the root into segments and distributes a different segment to process in the operational group (**132**). In scatter operation, all processes typically specify the same receive count. The send arguments are only significant to the root process, whose buffer actually contains sendcount*N elements of a given datatype, where N is the number of processes in the given group of compute nodes. The send buffer is divided and dispersed to all processes (including the process on the logical root). Each compute node is assigned a sequential identifier termed a 'rank.' After the operation, the root has sent sendcount data elements to each process in increasing rank order. Rank **0** receives the first sendcount data elements from the send buffer. Rank **1** receives the second sendcount data elements from the send buffer, and so on.

A gather operation is a many-to-one collective operation that is a complete reverse of the description of the scatter operation. That is, a gather is a many-to-one collective operation in which elements of a datatype are gathered from the ranked compute nodes into a receive buffer in a root node.

A reduction operation is also a many-to-one collective operation that includes an arithmetic or logical function performed on two data elements. All processes specify the same 'count' and the same arithmetic or logical function. After the reduction, all processes have sent count data elements from compute node send buffers to the root process. In a reduction operation, data elements from corresponding send buffer locations are combined pair-wise by arithmetic or logical operations to yield a single corresponding element in the root process' receive buffer. Application specific reduction operations can be defined at runtime. Parallel communications libraries may support predefined operations. MPI, for example, provides the following predefined reduction operations:

MPI_MAX	maximum
MPI_MIN	minimum
MPI_SUM	sum
MPI_PROD	product
MPI_LAND	logical and
MPI_BAND	bitwise and
MPI_LOR	logical or
MPI_BOR	bitwise or
MPI_LXOR	logical exclusive or
MPI_BXOR	bitwise exclusive or

In addition to compute nodes, the parallel computer (**100**) includes input/output ('I/O') nodes (**110**, **114**) coupled to compute nodes (**102**) through the global combining network (**106**). The compute nodes (**102**) in the parallel computer (**100**) may be partitioned into processing sets such that each compute node in a processing set is connected for data communications to the same I/O node. Each processing set, therefore, is composed of one I/O node and a subset of compute nodes (**102**). The ratio between the number of compute nodes to the number of I/O nodes in the entire system typically depends on the hardware configuration for the parallel computer (**102**). For example, in some configurations, each processing set may be composed of eight compute nodes and one I/O node. In some other configurations, each processing set may be composed of sixty-four compute nodes and one I/O node. Such example are for explanation only, however, and not for limitation. Each I/O node provides I/O services between compute nodes (**102**) of its processing set and a set of I/O devices. In the example of FIG. **1**, the I/O nodes (**110**, **114**) are connected for data communications I/O devices (**118**, **120**, **122**) through local area network ('LAN') (**130**) implemented using high-speed Ethernet.

The parallel computer (**100**) of FIG. **1** also includes a service node (**116**) coupled to the compute nodes through one of the networks (**104**). Service node (**116**) provides services common to pluralities of compute nodes, administering the configuration of compute nodes, loading programs into the compute nodes, starting program execution on the compute nodes, retrieving results of program operations on the compute nodes, and so on. Service node (**116**) runs a service application (**124**) and communicates with users (**128**) through a service application interface (**126**) that runs on computer terminal (**122**).

The parallel computer (**100**) of FIG. **1** operates generally for administering group identifiers of processes in a parallel computer in accordance with embodiments of the present invention. The processes of the compute nodes (**102**) of the example parallel computer (**100**) are organized in the operational group (**132**) for executing collective operation algorithms.

A collective operation algorithm is designed for multiple compute nodes to execute pieces of a program in parallel. Some collective operation algorithms are very efficient but may require a particular number of participants. Examples of such algorithms include algorithms that require a number of participants that are a power of two, a multiple of two, an odd number of participants, a prime number of participants and so on.

In the example parallel computer (**100**) of FIG. **1**, the compute nodes operate in accordance with embodiments of the present invention to administer group identifiers of processes in a parallel computer including each process in a set of processes, receiving from a compute node of the plurality of compute nodes, a request to establish the set of processes as an operational group. Receiving a request includes receiving a list of process identifiers for each process of the set of processes. Embodiments also include each process generating without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers.

Administering group identifiers of processes according to embodiments of the present invention is generally implemented on a parallel computer that includes a plurality of compute nodes organized for collective operations through at least one data communications network. In fact, such computers may include thousands of such compute nodes. Each compute node is in turn itself a kind of computer composed of one or more computer processing cores, its own computer memory, and its own input/output adapters. For further explanation, therefore, FIG. **2** sets forth a block diagram of an example compute node (**297**) useful in a parallel computer capable of administer group identifiers of processes according to embodiments of the present invention. The compute node (**297**) of FIG. **2** includes a plurality of processing cores (**165**) as well as RAM (**156**). The processing cores (**165**) of FIG. **2** may be configured on one or more integrated circuit dies. Processing cores (**165**) are connected to RAM (**156**) through a high-speed memory bus

(**155**) and through a bus adapter (**194**) and an extension bus (**168**) to other components of the compute node.

Stored in RAM (**156**) is an application program (**226**), a module of computer program instructions that carries out parallel, user-level data processing using parallel algorithms, such as collective operation algorithm (**228**). In the example of FIG. **2**, the application (**226**) may implement a participant in an operational group, such as a rank in an MPI-style communicator. Execution of the application program (**226**) causes the example compute node (**297**) of FIG. **2** to execute the collective operation algorithm (**228**) in accordance with embodiments of the present invention. The compute node (**297**) may carry out such collective operation algorithm execution by: a process (**298**) in a set of processes, receiving from a compute node of the plurality of compute nodes, a request to establish the set of processes as an operational group including receiving a list of process identifiers for each process of the set of processes. Embodiments also include the process (**298**) generating without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers.

Also stored RAM (**156**) is a parallel communications library (**161**), a library of computer program instructions that carry out parallel communications among compute nodes, including point-to-point operations as well as collective operations. A library of parallel communications routines may be developed from scratch for use in systems according to embodiments of the present invention, using a traditional programming language such as the C programming language, and using traditional programming methods to write parallel communications routines that send and receive data among nodes on two independent data communications networks. Alternatively, existing prior art libraries may be improved to operate according to embodiments of the present invention. Examples of prior-art parallel communications libraries include the 'Message Passing Interface' ('MPI') library and the 'Parallel Virtual Machine' ('PVM') library.

Also stored in RAM (**156**) is an operating system (**162**), a module of computer program instructions and routines for an application program's access to other resources of the compute node. It is typical for an application program and parallel communications library in a compute node of a parallel computer to run a single thread of execution with no user login and no security issues because the thread is entitled to complete access to all resources of the node. The quantity and complexity of tasks to be performed by an operating system on a compute node in a parallel computer therefore are smaller and less complex than those of an operating system on a serial computer with many threads running simultaneously. In addition, there is no video I/O on the compute node (**297**) of FIG. **2**, another factor that decreases the demands on the operating system. The operating system (**162**) may therefore be quite lightweight by comparison with operating systems of general purpose computers, a pared down version as it were, or an operating system developed specifically for operations on a particular parallel computer. Operating systems that may usefully be improved, simplified, for use in a compute node include UNIX™, Linux™, Windows XP™, AIX™, IBM's i5/OS™, and others as will occur to those of skill in the art.

The example compute node (**297**) of FIG. **2** includes several communications adapters (**172**, **176**, **180**, **188**) for implementing data communications with other nodes of a parallel computer. Such data communications may be carried out serially through RS-232 connections, through external buses such as USB, through data communications networks such as IP networks, and in other ways as will occur to those of skill in the art. Communications adapters implement the hardware level of data communications through which one computer sends data communications to another computer, directly or through a network. Examples of communications adapters useful in apparatus useful for administering group identifiers of processes in a parallel computer include modems for wired communications, Ethernet (IEEE 802.3) adapters for wired network communications, and 802.11b adapters for wireless network communications.

The data communications adapters in the example of FIG. **2** include a Gigabit Ethernet adapter (**172**) that couples example compute node (**297**) for data communications to a Gigabit Ethernet (**174**). Gigabit Ethernet is a network transmission standard, defined in the IEEE 802.3 standard, that provides a data rate of 1 billion bits per second (one gigabit). Gigabit Ethernet is a variant of Ethernet that operates over multimode fiber optic cable, single mode fiber optic cable, or unshielded twisted pair.

The data communications adapters in the example of FIG. **2** include a JTAG Slave circuit (**176**) that couples example compute node (**297**) for data communications to a JTAG Master circuit (**178**). JTAG is the usual name used for the IEEE 1149.1 standard entitled Standard Test Access Port and Boundary-Scan Architecture for test access ports used for testing printed circuit boards using boundary scan. JTAG is so widely adapted that, at this time, boundary scan is more or less synonymous with JTAG. JTAG is used not only for printed circuit boards, but also for conducting boundary scans of integrated circuits, and is also useful as a mechanism for debugging embedded systems, providing a convenient alternative access point into the system. The example compute node of FIG. **2** may be all three of these: It typically includes one or more integrated circuits installed on a printed circuit board and may be implemented as an embedded system having its own processing core, its own memory, and its own I/O capability. JTAG boundary scans through JTAG Slave (**176**) may efficiently configure processing core registers and memory in compute node (**297**) for use in dynamically reassigning a connected node to a block of compute nodes useful in systems for administering group identifiers of processes in a parallel computer to embodiments of the present invention.

The data communications adapters in the example of FIG. **2** include a Point-To-Point Network Adapter (**180**) that couples example compute node (**297**) for data communications to a network (**108**) that is optimal for point-to-point message passing operations such as, for example, a network configured as a three-dimensional torus or mesh. The Point-To-Point Adapter (**180**) provides data communications in six directions on three communications axes, x, y, and z, through six bidirectional links: +x (**181**), −x (**182**), +y (**183**), −y (**184**), +z (**185**), and −z (**186**).

The data communications adapters in the example of FIG. **2** include a Global Combining Network Adapter (**188**) that couples example compute node (**297**) for data communications to a global combining network (**106**) that is optimal for collective message passing operations such as, for example, a network configured as a binary tree. The Global Combining Network Adapter (**188**) provides data communications through three bidirectional links for each global combining network (**106**) that the Global Combining Network Adapter (**188**) supports. In the example of FIG. **2**, the Global Combining Network Adapter (**188**) provides data communications through three bidirectional links for global combining network (**106**): two to children nodes (**190**) and one to a parent node (**192**).

The example compute node (**297**) includes multiple arithmetic logic units ('ALUs'). Each processing core (**165**) includes an ALU (**166**), and a separate ALU (**170**) is dedicated to the exclusive use of the Global Combining Network Adapter (**188**) for use in performing the arithmetic and logical functions of reduction operations, including an allreduce operation. Computer program instructions of a reduction routine in a parallel communications library (**161**) may latch an instruction for an arithmetic or logical function into an instruction register (**169**). When the arithmetic or logical function of a reduction operation is a 'sum' or a 'logical OR,' for example, the collective operations adapter (**188**) may execute the arithmetic or logical operation by use of the ALU (**166**) in the processing core (**165**) or, typically much faster, by use of the dedicated ALU (**170**) using data provided by the nodes (**190**, **192**) on the global combining network (**106**) and data provided by processing cores (**165**) on the compute node (**297**).

Often when performing arithmetic operations in the global combining network adapter (**188**), however, the global combining network adapter (**188**) only serves to combine data received from the children nodes (**190**) and pass the result up the network (**106**) to the parent node (**192**). Similarly, the global combining network adapter (**188**) may only serve to transmit data received from the parent node (**192**) and pass the data down the network (**106**) to the children nodes (**190**). That is, none of the processing cores (**165**) on the compute node (**297**) contribute data that alters the output of ALU (**170**), which is then passed up or down the global combining network (**106**). Because the ALU (**170**) typically does not output any data onto the network (**106**) until the ALU (**170**) receives input from one of the processing cores (**165**), a processing core (**165**) may inject the identity element into the dedicated ALU (**170**) for the particular arithmetic operation being perform in the ALU (**170**) in order to prevent alteration of the output of the ALU (**170**). Injecting the identity element into the ALU, however, often consumes numerous processing cycles. To further enhance performance in such cases, the example compute node (**297**) includes dedicated hardware (**171**) for injecting identity elements into the ALU (**170**) to reduce the amount of processing core resources required to prevent alteration of the ALU output. The dedicated hardware (**171**) injects an identity element that corresponds to the particular arithmetic operation performed by the ALU. For example, when the global combining network adapter (**188**) performs a bitwise OR on the data received from the children nodes (**190**), dedicated hardware (**171**) may inject zeros into the ALU (**170**) to improve performance throughout the global combining network (**106**).

For further explanation, FIG. **3**A sets forth a block diagram of an example Point-To-Point Adapter (**180**) useful in systems for administering group identifiers of processes in a parallel computer according to embodiments of the present invention. The Point-To-Point Adapter (**180**) is designed for use in a data communications network optimized for point-to-point operations, a network that organizes compute nodes in a three-dimensional torus or mesh. The Point-To-Point Adapter (**180**) in the example of FIG. **3**A provides data communication along an x-axis through four unidirectional data communications links, to and from the next node in the −x direction (**182**) and to and from the next node in the +x direction (**181**). The Point-To-Point Adapter (**180**) of FIG. **3**A also provides data communication along a y-axis through four unidirectional data communications links, to and from the next node in the −y direction (**184**) and to and from the next node in the +y direction (**183**). The Point-To-Point Adapter (**180**) of FIG. **3**A also provides data communication along a z-axis through four unidirectional data communications links, to and from the next node in the −z direction (**186**) and to and from the next node in the +z direction (**185**).

For further explanation, FIG. **3**B sets forth a block diagram of an example Global Combining Network Adapter (**188**) useful in systems for administering group identifiers of processes in a parallel computer according to embodiments of the present invention. The Global Combining Network Adapter (**188**) is designed for use in a network optimized for collective operations, a network that organizes compute nodes of a parallel computer in a binary tree. The Global Combining Network Adapter (**188**) in the example of FIG. **3**B provides data communication to and from children nodes of a global combining network through four unidirectional data communications links (**190**), and also provides data communication to and from a parent node of the global combining network through two unidirectional data communications links (**192**).

For further explanation, FIG. **4** sets forth a line drawing illustrating an example data communications network (**108**) optimized for point-to-point operations useful in systems capable of administering group identifiers of processes in a parallel computer according to embodiments of the present invention. In the example of FIG. **4**, dots represent compute nodes (**102**) of a parallel computer, and the dotted lines between the dots represent data communications links (**103**) between compute nodes. The data communications links are implemented with point-to-point data communications adapters similar to the one illustrated for example in FIG. **3**A, with data communications links on three axis, x, y, and z, and to and fro in six directions +x (**181**), −x (**182**), +y (**183**), −y (**184**), +z (**185**), and −z (**186**). The links and compute nodes are organized by this data communications network optimized for point-to-point operations into a three dimensional mesh (**105**). The mesh (**105**) has wrap-around links on each axis that connect the outermost compute nodes in the mesh (**105**) on opposite sides of the mesh (**105**). These wrap-around links form a torus (**107**). Each compute node in the torus has a location in the torus that is uniquely specified by a set of x, y, z coordinates. Readers will note that the wrap-around links in the y and z directions have been omitted for clarity, but are configured in a similar manner to the wrap-around link illustrated in the x direction. For clarity of explanation, the data communications network of FIG. **4** is illustrated with only 27 compute nodes, but readers will recognize that a data communications network optimized for point-to-point operations for use in administering group identifiers of processes in a parallel computer in accordance with embodiments of the present invention may contain only a few compute nodes or may contain thousands of compute nodes. For ease of explanation, the data communications network of FIG. **4** is illustrated with only three dimensions, but readers will recognize that a data communications network optimized for point-to-point operations for use in administering group identifiers of processes in a parallel computer in accordance with embodiments of the present invention may in fact be implemented in two dimensions, four dimensions, five dimensions, and so on. Several supercomputers now use five dimensional mesh or torus networks, including, for example, IBM's Blue Gene Q™.

For further explanation, FIG. **5** sets forth a line drawing illustrating an example global combining network (**106**) useful in systems capable of administering group identifiers of processes in a parallel computer according to embodiments of the present invention. The example data communications network of FIG. **5** includes data communications links (**103**) connected to the compute nodes so as to organize the compute nodes as a tree. In the example of FIG. **5**, dots represent compute nodes (**102**) of a parallel computer, and the dotted lines (**103**) between the dots represent data communications links between compute nodes. The data communications links are implemented with global combining network adapters similar to the one illustrated for example in FIG. **3**B, with each node typically providing data communications to and from two children nodes and data communications to and from a parent node, with some exceptions. Nodes in the global combining network (**106**) may be characterized as a physical root node (**202**), branch nodes (**204**), and leaf nodes (**206**). The physical root (**202**) has two children but no parent and is so called because the physical root node (**202**) is the node physically configured at the top of the binary tree. The leaf nodes (**206**) each has a parent, but leaf nodes have no children. The branch nodes (**204**) each has both a parent and two children. The links and compute nodes are thereby organized by this data communications network optimized for collective operations into a binary tree (**106**). For clarity of explanation, the data communications network of FIG. **5** is illustrated with only 31 compute nodes, but readers will recognize that a global combining network (**106**) optimized for collective operations for use in administering group identifiers of processes in a parallel computer in accordance with embodiments of the present invention may contain only a few compute nodes or may contain thousands of compute nodes.

In the example of FIG. **5**, each node in the tree is assigned a unit identifier referred to as a 'rank' (**250**). The rank actually identifies a task or process that is executing a parallel operation according to embodiments of the present invention. Using the rank to identify a node assumes that only one such task is executing on each node. To the extent that more than one participating task executes on a single node, the rank identifies the task as such rather than the node. A rank uniquely identifies a task's location in the tree network for use in both point-to-point and collective operations in the tree network. The ranks in this example are assigned as integers beginning with 0 assigned to the root tasks or root node (**202**), 1 assigned to the first node in the second layer of the tree, 2 assigned to the second node in the second layer of the tree, 3 assigned to the first node in the third layer of the tree, 4 assigned to the second node in the third layer of the tree, and so on. For ease of illustration, only the ranks of the first three layers of the tree are shown here, but all compute nodes in the tree network are assigned a unique rank.

For further explanation, FIG. **6** sets forth a flow chart illustrating an example method administering group identifiers of processes in a parallel computer according to embodiments of the present invention. The method of FIG. **6** includes each process (**662**, **664**, **667**) in a set (**660**) of processes receiving (**602**) from a compute node (**680**), a request (**650**) to establish the set (**660**) of processes as an operational group. A process is an execution path through address space—in other words, a set of computer program instructions that are loaded in memory. In a High Performance Computing (HPC) environment, large computing processes of an application may be split up and specially adapted to execute collective parallel operations on multiple processors.

Each process (**662**, **664**, **667**) of FIG. **6** executes on one or more of compute nodes of a parallel computer, such as the compute nodes (**102**) of FIG. **1**. The processes (**662**, **664**, **667**) may be organized into an operational group for executing collective parallel operations. Each operational group of processes is the set of processes that execute a collective parallel operation. Each process in the operational group is assigned a unique rank or process identifier that identifies the particular process in the operational group.

Collective operations are implemented with data communications among the processes of an operational group. Collective operations are those functions that involve all the processes of an operational group. A collective operation is an operation, a message-passing computer program instruction that is executed simultaneously, that is, at approximately the same time, by all the processes in an operational group of processes. Such an operational group may include all the processes executing on the compute nodes in a parallel computer or a subset of all the processes executing on the compute nodes. Collective operations are often built around point-to-point operations. A collective operation requires that all processes within an operational group call the same collective operation with matching arguments. A 'broadcast' is an example of a collective operation for moving data among processes of an operational group. A 'reduce' operation is an example of a collective operation that executes arithmetic or logical functions on data distributed among the processes of an operational group. An operational group may be implemented as, for example, an MPI 'communicator.'

Receiving (**602**) from a compute node (**680**), a request (**650**) to establish the set (**660**) of processes as an operational group may be carried out by receiving a message that indicates a command to create an operational group based on one or more other process identifiers in addition to the process identifier of the process receiving the request.

In the method of FIG. **6**, receiving (**602**) a request (**650**) to establish the set (**660**) of processes as an operational group includes each process (**662**, **664**, **667**) in a set (**660**) of processes receiving (**604**) a list (**652**) of process identifiers for each process (**662**, **664**, **667**) of the set (**660**) of processes. Receiving (**604**) a list (**652**) of process identifiers for each process (**662**, **664**, **667**) of the set (**660**) of processes may be carried out by receiving an operational group establishment request that includes indications of the process identifiers for each process to be included in the operational group.

The method of FIG. **6** includes each process (**662**, **664**, **667**) in a set (**660**) of processes includes generating (**606**) without communication amongst the processes (**662**, **664**, **667**), a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers. Generating (**606**) without communication amongst the processes (**662**, **664**, **667**), a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers may be carried out by applying a group identifier creation algorithm to the actual process identifiers to create a group identifier. That is, each process, independently and locally creates the same group identifier. Because each process is able to independently create the same group identifier, the processes of the operational group do not need to communicate with each other to establish and agree on the group identifier. Eliminating the need for communication between the processes for establishment of the group identifiers also eliminates the need for a central location to manage group identifiers and un-necessary traffic to distribute those group identifiers.

For further explanation, FIG. **7** sets forth a flow chart illustrating another example method administering group identifiers of processes in a parallel computer according to embodiments of the present invention. The method of FIG. **7** is similar to the method of FIG. **6** in that the method of

FIG. **7** also includes receiving (**602**) from a compute node (**680**) of the plurality of compute nodes, by each process (**662**, **664**, **667**) in the set (**660**) of processes, a request (**650**) to establish the set (**660**) of processes as an operational group including receiving (**604**) a list (**652**) of process identifiers for each process (**662**, **664**, **667**) of the set (**660**) of processes; and generating (**606**), by each process (**662**, **664**, **667**) without communication amongst the processes (**662**, **664**, **667**), a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers.

In the method of FIG. **7**, however, generating (**606**) a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers includes each process (**662**, **664**, **667**) generating (**702**) based on the list (**652**) of process identifiers a stride triplet (**750**) comprising the following elements: a lowest process identifier (**790**), a value (**792**) representing a stride size between the set (**660**) of processes, and a number (**794**) of processes in the set (**660**) of processes. Generating (**702**) based on the list (**652**) of process identifiers a stride triplet (**750**) may be carried out by identifying the process identifier having the lowest number of the process identifiers; determining the number between any two process identifiers of the set of process identifiers; and the number of processes in the set of processes.

In the method of FIG. **7**, generating (**606**) a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers includes each process (**662**, **664**, **667**) generating (**704**) without communication amongst the processes (**662**, **664**, **667**) and in dependence upon the stride triplet (**750**), the unique group identifier (**670**). Generating (**704**) without communication amongst the processes (**662**, **664**, **667**) and in dependence upon the stride triplet (**750**), the unique group identifier (**670**) may be carried out by combining the elements of the stride triplet together; or inputting one or more elements of the stride triplet into a function that outputs the unique group identifier.

In the method of FIG. **7**, generating (**704**) without communication amongst the processes (**662**, **664**, **667**) and in dependence upon the stride triplet (**750**), the unique group identifier (**670**) includes concatenating (**706**) the elements (**790**, **792**, **794**) of the stride triplet (**750**). Concatenating (**706**) the elements (**790**, **792**, **794**) of the stride triplet (**750**) may be carried out by joining the lowest process identifier (**790**), a value (**792**) representing a stride size between the set (**660**) of processes, and a number (**794**) of processes in the set (**660**) of processes together to form the unique group identifier.

For further explanation, FIG. **8** sets forth a flow chart illustrating another example method administering group identifiers of processes in a parallel computer according to embodiments of the present invention. The method of FIG. **8** is similar to the method of FIG. **6** in that the method of FIG. **8** also includes receiving (**602**) from a compute node (**680**) of the plurality of compute nodes, by each process (**662**, **664**, **667**) in the set (**660**) of processes, a request (**650**) to establish the set (**660**) of processes as an operational group including receiving (**604**) a list (**652**) of process identifiers for each process (**662**, **664**, **667**) of the set (**660**) of processes; and generating (**606**), by each process (**662**, **664**, **667**) without communication amongst the processes (**662**, **664**, **667**), a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers. The method of FIG. **8** is similar to the method of FIG. **7** in that the method of FIG. **8** also includes generating (**702**) based on the list (**652**) of process identifiers a stride triplet (**750**); and generating (**704**) without communication amongst the processes (**662**, **664**, **667**) and in dependence upon the stride triplet (**750**), the unique group identifier (**670**).

The method of FIG. **8** also includes each process (**662**, **664**, **667**) receiving (**802**) a second request (**850**) to establish the set (**660**) of processes as a second group. Receiving (**802**) a second request (**850**) to establish the set (**660**) of processes as a second group may be carried out by receiving a message that indicates a command to create an operational group based on one or more other process identifiers in addition to the process identifier of the process receiving the request.

The method of FIG. **8** also includes each process (**662**, **664**, **667**), responsive to receiving the second request (**850**), without previously destroying the operational group, generating (**804**) a second unique group identifier (**890**) for the second group in dependence upon the stride triplet (**750**) and an instance identifier (**852**). Generating (**804**) a second unique group identifier (**890**) for the second group in dependence upon the stride triplet (**750**) and an instance identifier (**852**) may be carried out by using both the stride triplet and the instance identifier to generate a new group identifier. For example, half of the bits of the group identifier may be maintained and the other half may be comprised of the instance identifier.

In the method of FIG. **8**, generating (**804**) a second unique group identifier (**890**) for the second group in dependence upon the stride triplet (**750**) and an instance identifier (**852**) includes appending (**806**) the instance identifier (**852**) to the stride triplet (**750**). Appending (**806**) the instance identifier (**852**) to the stride triplet (**750**) may be carried out by combining in some way the instance identifier to the stride triplet, such as concatenating or using a hash function to modify the stride triplet.

For further explanation, FIG. **9** sets forth a flow chart illustrating another example method administering group identifiers of processes in a parallel computer according to embodiments of the present invention. The method of FIG. **9** is similar to the method of FIG. **6** in that the method of FIG. **9** also includes receiving (**602**) from a compute node (**680**) of the plurality of compute nodes, by each process (**662**, **664**, **667**) in the set (**660**) of processes, a request (**650**) to establish the set (**660**) of processes as an operational group including receiving (**604**) a list (**652**) of process identifiers for each process (**662**, **664**, **667**) of the set (**660**) of processes; and generating (**606**), by each process (**662**, **664**, **667**) without communication amongst the processes (**662**, **664**, **667**), a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers.

In the method of FIG. **9**, generating (**606**) a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers includes each process (**662**, **664**, **667**) sorting (**902**) the process identifiers of the list (**652**) of process identifiers according to a predefined sort criteria (**950**). Sorting (**902**) the process identifiers of the list (**652**) of process identifiers according to a predefined sort criteria (**950**) may be carried out by arranging the process identifiers by a particular algorithm such that each process generates the same sorted process identifiers (**952**).

In the method of FIG. **9**, generating (**606**) a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers includes each process (**662**, **664**, **667**) concatenating (**904**) the sorted process identifiers (**952**) to generate concatenated sorted process identifiers (**956**). In the example of FIG. **9**, the concatenated process identifiers (**956**) comprise the unique group identifier (**670**). Concatenating (**904**) the sorted process identifiers (**952**) to generate concatenated sorted process identifiers (**956**) may be carried out by joining each of the process identifiers together to form the group identifier.

For further explanation, FIG. **10** sets forth a flow chart illustrating another example method administering group identifiers of processes in a parallel computer according to embodiments of the present invention. The method of FIG. **10** is similar to the method of FIG. **6** in that the method of FIG. **10** also includes receiving (**602**) from a compute node (**680**) of the plurality of compute nodes, by each process (**662**, **664**, **667**) in the set (**660**) of processes, a request (**650**) to establish the set (**660**) of processes as an operational group including receiving (**604**) a list (**652**) of process identifiers for each process (**662**, **664**, **667**) of the set (**660**) of processes; and generating (**606**), by each process (**662**, **664**, **667**) without communication amongst the processes (**662**, **664**, **667**), a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers.

In the method of FIG. **10**, generating (**606**) a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers includes each process (**662**, **664**, **667**) sorting (**1002**) the process identifiers of the list (**652**) of process identifiers according to a predefined sort criteria (**1050**). Sorting (**1002**) the process identifiers of the list (**652**) of process identifiers according to a predefined sort criteria (**1050**) may be carried out by arranging the process identifiers by a particular algorithm such that each process generates the same sorted process identifiers (**1052**).

In the method of FIG. **10**, generating (**606**) a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers includes each process (**662**, **664**, **667**) hashing (**1004**) the sorted process identifiers (**1052**) to generate a hash value (**1054**). In the example of FIG. **10**, the hash value (**1054**) comprises the unique group identifier (**670**). Hashing (**1004**) the sorted process identifiers (**1052**) to generate a hash value (**1054**) may be carried out by applying a hash function to generate output data.

In a particular embodiment, hash collision may occur when one process, which is a member of a second operational group, is also a member of a first operational group that is trying to generate a unique identifier. The unique identifier of the second operational group may—for many different reasons—be the same hash value generated by the processes of the first operational group. For example, the process (**662**) may be a member of a second operational group and a member of the first operational group. In this example, the hash value (**1054**) matches a second group identifier (**1058**) of the second operational group.

The method of FIG. **10** includes the process (**662**) determining (**1005**) that the group identifier (**1058**) of the second operational group matches the hash value (**1054**). Determining (**1005**) that the group identifier (**1058**) of the second operational group matches the hash value (**1054**) may be carried out by comparing the hash value to the group identifier (**1058**) of the second operational group.

In response to determining that that group identifier (**1058**) of the second operational group matches the hash value (**1054**), the method of FIG. **10** continues by the at least one process, process (**622**) in this example, informing the other processes (**664**, **667**) of the first operational group, that the group identifier (**1058**) of the second operational group matches the hash value (**1054**). Informing the other processes (**664**, **667**) of the first operational group, that the group identifier (**1058**) of the second operational group matches the hash value (**1054**) may be carried out by transmitting a message to each of the processes of the first operational group.

The method of FIG. **10** includes each process (**662**, **664**, **667**), in response to the group identifier (**1058**) of the second operational group matching the hash value (**1054**), changing (**1010**) the hash value (**1054**) to a same new hash value (**1060**). Changing (**1010**) the hash value (**1054**) to a same new hash value (**1060**) may be carried out by increasing the hash value (**1054**) by a predefined amount to generate the increased hash value (**1060**). That is, each process increases the hash value by the same amount to generate the new hash value to use to create the same group identifier for the first operational group. This method of identifying a hash value conflict with a group identifier of another operational group, informing the other processes of the operational group, and changing the hash value may be performed iteratively until each process does not determine that the hash value conflicts with any group identifiers.

For further explanation, FIG. **11** sets forth a flow chart illustrating another example method administering group identifiers of processes in a parallel computer according to embodiments of the present invention. The method of FIG. **11** is similar to the method of FIG. **6** in that the method of FIG. **11** also includes receiving (**602**) from a compute node (**680**) of the plurality of compute nodes, by each process (**662**, **664**, **667**) in the set (**660**) of processes, a request (**650**) to establish the set (**660**) of processes as an operational group including receiving (**604**) a list (**652**) of process identifiers for each process (**662**, **664**, **667**) of the set (**660**) of processes; and generating (**606**), by each process (**662**, **664**, **667**) without communication amongst the processes (**662**, **664**, **667**), a unique group identifier (**670**) in dependence upon the list (**652**) of process identifiers.

The method of FIG. **11** includes each process (**662**, **664**, **667**) destroying (**1102**) the unique group identifier (**670**) in response to destruction of the operational group. An operational group may be destroyed as a result of completion of the parallel collective operations amongst the processes. Destroying (**1102**) the unique group identifier (**670**) in response to destruction of the operational group may be carried out by each process locally deleting data indicating the unique group identifier.

As will be appreciated by one skilled in the art, aspects of the present invention may be embodied as a system, method or computer program product. Accordingly, aspects of the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "circuit," "module" or "system." Furthermore, aspects of the present invention may take the form of a computer program product embodied in one or more computer readable medium(s) having computer readable program code embodied thereon.

Any combination of one or more computer readable medium(s) may be utilized. The computer readable medium may be a computer readable transmission medium or a computer readable storage medium. A computer readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. More specific examples (a non-exhaustive list) of the computer readable storage medium would include the following: an electrical connection having one or more wires, a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device.

A computer readable transmission medium may include a propagated data signal with computer readable program code embodied therein, for example, in baseband or as part of a carrier wave. Such a propagated signal may take any of a variety of forms, including, but not limited to, electro-magnetic, optical, or any suitable combination thereof. A computer readable transmission medium may be any computer readable medium that is not a computer readable storage medium and that can communicate, propagate, or transport a program for use by or in connection with an instruction execution system, apparatus, or device.

Program code embodied on a computer readable medium may be transmitted using any appropriate medium, including but not limited to wireless, wireline, optical fiber cable, RF, etc., or any suitable combination of the foregoing.

Computer program code for carrying out operations for aspects of the present invention may be written in any combination of one or more programming languages, including an object oriented programming language such as Java, Smalltalk, C++ or the like and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The program code may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider).

Aspects of the present invention are described above with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer readable medium that can direct a computer, other programmable data processing apparatus, or other devices to function in a particular manner, such that the instructions stored in the computer readable medium produce an article of manufacture including instructions which implement the function/act specified in the flowchart and/or block diagram block or blocks.

The computer program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other devices to cause a series of operational steps to be performed on the computer, other programmable apparatus or other devices to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

It will be understood from the foregoing description that modifications and changes may be made in various embodiments of the present invention without departing from its true spirit. The descriptions in this specification are for purposes of illustration only and are not to be construed in a limiting sense. The scope of the present invention is limited only by the language of the following claims.

What is claimed is:

1. A method of administering group identifiers of processes in a parallel computer, the parallel computer comprising a plurality of compute nodes, the compute nodes coupled for data communications by one or more data communications networks, one or more of the compute nodes executing a set of processes, the set of processes organized in an operational group for collective parallel operations, the method comprising:

receiving from a compute node of the plurality of compute nodes, by each process in the set of processes, a request to establish the set of processes as an operational group including receiving a list of process identifiers for each process of the set of processes; and

generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers.

2. The method of claim **1** wherein generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers includes:

generating based on the list of process identifiers, by each process, a stride triplet comprising the following elements: a lowest process identifier, a value representing a stride size between the set of processes, and a number of processes in the set of processes; and

generating, by each process without communication amongst the processes and in dependence upon the stride triplet, the unique group identifier.

3. The method of claim **2** wherein generating, by each process in dependence upon the stride triplet, a unique group identifier further comprises:

concatenating the elements of the stride triplet.

4. The method of claim **2**, further comprising:

responsive to receiving a second request to establish the set of processes as a second group, without previously destroying the operational group, generating a second unique group identifier for the second group in dependence upon the stride triplet and an instance identifier.

5. The method of claim **4** wherein generating a second unique group identifier for the second group in dependence upon the stride triplet and an instance identifier includes appending the instance identifier to the stride triplet.

6. The method of claim **1** wherein generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers further comprises:

sorting the process identifiers of the list of process identifiers according to a predefined sort criteria; and

concatenating the sorted process identifiers, wherein the concatenated process identifiers comprises the unique group identifier.

7. The method of claim **1** wherein generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers further comprises:

sorting the process identifiers of the list of process identifiers according to a predefined sort criteria; and

hashing the sorted process identifiers to generate a hash value, wherein the hash value comprises the unique group identifier.

8. The method of claim **7** wherein

at least one process is a member of a second operational group and the group identifier of the second operational group matches the hash value; and

wherein the method further includes:

determining, by the at least one process, that the group identifier of the second operational group matches the hash value; and

in response to determining that the group identifier of the second operational group matches the hash value, informing, by the at least one process, the other processes of the operational group, that the group identifier of the second operational group matches the hash value; and

in response to the group identifier of the second operational group matching the hash value, changing, by each process, the hash value to a same new hash value.

9. The method of claim **1**, further comprising destroying, by each process, the unique group identifier in response to destruction of the operational group.

10. An apparatus for executing administering group identifiers of processes in a parallel computer, the parallel computer comprising a plurality of compute nodes, the compute nodes coupled for data communications by one or more data communications networks, one or more of the compute nodes executing a set of processes, the set of processes organized in an operational group for collective parallel operations, the apparatus comprising a computer processor and computer memory operatively coupled to the computer processor, the computer memory having disposed within it computer program instructions that, when executed by the computer processor, cause the apparatus to carry out the steps of:

receiving from a compute node of the plurality of compute nodes, by each process in the set of processes, a request to establish the set of processes as an operational group including receiving a list of process identifiers for each process of the set of processes; and

generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers.

11. The apparatus of claim **10** wherein generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers includes:

generating based on the list of process identifiers, by each process, a stride triplet comprising the following elements: a lowest process identifier, a value representing a stride size between the set of processes, and a number of processes in the set of processes; and

generating, by each process without communication amongst the processes and in dependence upon the stride triplet, the unique group identifier.

12. The apparatus of claim **10** wherein generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers further comprises:

sorting the process identifiers of the list of process identifiers according to a predefined sort criteria; and

concatenating the sorted process identifiers, wherein the concatenated process identifiers comprises the unique group identifier.

13. The apparatus of claim **10** wherein generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers further comprises:

sorting the process identifiers of the list of process identifiers according to a predefined sort criteria; and

hashing the sorted process identifiers to generate a hash value, wherein the hash value comprises the unique group identifier.

14. A computer program product for administering group identifiers of processes in a parallel computer, the parallel computer comprising a plurality of compute nodes, the compute nodes coupled for data communications by one or more data communications networks, one or more of the compute nodes executing a set of processes, the set of processes organized in an operational group for collective parallel operations, the computer program product disposed upon a computer readable medium, the computer program product comprising computer program instructions that, when executed, cause a computer to carry out the steps of:

receiving from a compute node of the plurality of compute nodes, by each process in the set of processes, a request to establish the set of processes as an operational group including receiving a list of process identifiers for each process of the set of processes; and

generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers.

15. The computer program product of claim **14** wherein generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers includes:

generating based on the list of process identifiers, by each process, a stride triplet comprising the following elements: a lowest process identifier, a value representing a stride size between the set of processes, and a number of processes in the set of processes; and

generating, by each process without communication amongst the processes and in dependence upon the stride triplet, the unique group identifier.

16. The computer program product of claim **14** wherein generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers further comprises:

sorting the process identifiers of the list of process identifiers according to a predefined sort criteria; and

concatenating the sorted process identifiers, wherein the concatenated process identifiers comprises the unique group identifier.

17. The computer program product of claim **14** wherein generating, by each process without communication amongst the processes, a unique group identifier in dependence upon the list of process identifiers further comprises:

sorting the process identifiers of the list of process identifiers according to a predefined sort criteria; and

hashing the sorted process identifiers to generate a hash value, wherein the hash value comprises the unique group identifier.

18. The computer program product of claim **14**, further comprising:

destroying, by each process, the unique group identifier in response to destruction of the operational group.

19. The computer program product of claim **14** wherein the computer readable medium comprises a signal medium.

20. The computer program product of claim **14** wherein the computer readable medium comprises a storage medium.

* * * * *